COP 5536 Fall 2018

Aditi Malladi
aditi.malladi@ufl.edu
UFID: 9828-6321

Project Aim:
To be able to find out the most popular *n* keywords from a given input(read from a input file).
Input is to be read from a file and a *Max Fibonacci Heap* is to keep count of the keywords and have a pointer to the most popular keyword.
Output is then written into the output file. Each line has an output for a query.

Project Code:
There are 3 classes included for the project:

1. Node.java
2. FibonacciHeap.java
3. keywordcounter.java

Node.java
This class is used to define the structure and the values that need to be stored in each node.
Each node has the following data elements:

1. Node left, right, parent, child : References to nodes in the node-level circularly linked list
2. int degree : number of children node has (initially 0)
3. boolean childCut : if the node lost any of its children (initially 0)
4. int count : int count represents the number of times keyword appeared so far
5. String keyword : which is the string keyword

Node(String keyword, int count)

Description	Constructor to initialize new Node
Parameters	keyword : which is the string keyword count : int count represents the number of times keyword appeared so far
Return Value	-

FibonacciHeap.java
This class defines the various functions for a MAX fibonacci heap.
Has a **root** that points to the the max value of the heap.

This class defines various functions to perform the fibonacci heap operations.

void insert(Node node);

Description	To insert a node into the top level circular linked list
Parameters	Node node - node that needs to be inserted into the top level circular linked list
Return Value	-

void cascadingCut(Node node);

Description	To check if the a node has lost a child before, if yes then the node is leveled up to the top-level circular linked list level It is recursively called till a childCut value of false or a root node(part of top-level circular linked list) is encountered
Parameters	Node node - node that needs to be inserted into the top level circular linked list
Return Value	-

void addChild(Node x, Node y);
This function sets x as the parent of the node y
If x already has children, y is added to the sibling circular linked list.

Description	Making Node y a child node of Node x
Parameters	Node x - parent node Node y - child node
Return Value	-

void meld(Node temp, HashMap<Integer, Node>map);

Description	Melding trees according to the degree. (Ensuring only one tree of a given degree can exist in the map) It is a recursive function
Parameters	Node temp - the Node that meld function needs to be called upon (begins with root node) HashMap<Integer, Node>map - stores the degree value and the reference to the node that has that degree
Return Value	-

void resetRootPointer();

Description	Resets the root pointer to point to the maximum value
Parameters	-
Return Value	-

Node removeMax();

Description	To remove the root node from the heap(which points to the max value in the top level circular linked list)
Parameters	-
Return Value	Node root - returns the root node that has been removed

Node remove(Node node);

Description	Function to remove a node as a child from it’s parent node
Parameters	Node node - the node that needs to be removed from it’s parent
Return Value	Node node - returns the node that has been removed

void increaseKey(Node node, int incr);

Description	To increase the value of count for a given node by some amount

	(Also checks if the child count is greater than the parent and removes the node accordingly)
Parameters	Node node - The node whose count needs to be incremented int incr - the amount by which the node's count needs to be increased by
Return Value	-

keywordcounter.java

keywordcounter.java includes main() function

static void main(String args[])

Description	Main file for popularKeywords Read the input from the input file (path and name passed as a parameter)
Parameters	String args[] - file path with name is passed
Return Value	- (Function writes output into the output_file.txt)

Conclusion:

Using the FibonacciHeap we can keep track of the keyword that has the max count and easily get the one that has the maximum value with O(1) complexity.

This helps us easily get the top nodes.